VLDBJ manuscript No.
(will be inserted by the editor)

Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies

Sergio Mascetti · Dario Freni · Claudio Bettini · X. Sean Wang · Sushil Jajodia

Received: date / Accepted: date

Abstract A major feature of the emerging geo-social networks is the ability to notify a user when one of his friends (also called buddies) happens to be geographically in proximity with the user. This *proximity service* is usually offered by the network itself or by a third party service provider (SP) using location data acquired from the users. This paper provides a rigorous theoretical and experimental analysis of the existing solutions for the location privacy problem in proximity services. This is a serious problem for users who do not trust the SP to handle their location data, and would only like to release their location information in a generalized form to participating buddies. The paper presents two new protocols providing complete privacy with respect to the SP, and controllable privacy with respect to the buddies. The analytical and experimental analysis of the protocols takes into account privacy, service precision, and computation and communication costs, showing the superiority of the new protocols compared to those appeared in the literature to date. The proposed protocols have also been tested in a full system implementation of the proximity service.

Keywords Proximity services, geo-social networks, location-based services, location privacy

S. Mascetti, D. Freni, and C. Bettini
DICo, Università degli Studi di Milano
E-mail: {mascetti,freni,bettini}@dico.unimi.it

X.S. Wang
Department of CS, University of Vermont
E-mail: xywang@cs.uvm.edu

S. Jajodia
CSIS, George Mason University
E-mail: jajodia@gmu.edu

1 Introduction

A geo-social network is an extension of a social network in which the geographical positions of participants and of relevant resources are used to enable new information services. These networks are mostly motivated by the increased availability of GPS-enabled mobile devices that support both location-based services, and easy access to the current social networks.

As in most social networks, each user has a contact list of *friends*, also called *buddies*. A basic service in geo-social networks is the *proximity service* that alerts the user when any of her buddies is in the vicinity, possibly enacting other activities like visualizing the buddy's position on a map, or activating a communication session with the buddy. Such proximity services, often called *friend finder*, are already available as part of geo-social networks (e.g., *Brightkite*[1]), as part of a suite of map and navigation services (e.g., *Google Latitude*[2]), or as an independent service that can be integrated with social networks (e.g., *Loopt*[3]).

From a data management point of view, a proximity service involves the computation of a range query over a set of moving entities issued by a moving user, where the range is a distance threshold value decided by the user. All existing services are based on a centralized architecture in which location updates, issued from mobile devices[4], are acquired by the SP, and proximity is computed based on the acquired locations.

[1] `http://brightkite.com`

[2] `http://www.google.com/latitude`

[3] `http://www.loopt.com`

[4] While a variety of positioning technologies and communication infrastructure can be used, here we assume GPS-enabled devices with always-on 3G data connection.

The privacy problem

While proximity services are very attractive for many social network users, they also raise severe privacy concerns: a) the users may not fully trust the service provider that will handle their location data, b) the users would like to have better control on the precision of location data released to their buddies. For the purpose of alleviating these concerns, in this paper we address the problem of protecting users' location privacy in the context of proximity services, considering different possible *adversaries*, including the SP and, to a different extent, the buddies.

Existing proximity services do not offer any protection regarding point a) above other than legal privacy policy statements, and they offer a very limited control regarding point b); for example, some solutions allow the user to limit the location released to the buddies to the precision level of city. Point a) above has been addressed as part of a recent research effort on this topic by proposing a decentralized (peer-to-peer) architecture, and a number of protocols that compute proximity without the intervention of the SP [19].

Despite the smart protocols being proposed, we argue that a centralized or SP-mediated architecture, in addition to support current business models, may be more appropriate for a proximity service than a decentralized architecture, since reducing the communication and computation costs on the clients is often a primary goal. A centralized architecture imposes either a complete trust of the user on the central entity, i.e., the SP, or the need for a protocol that is proved to limit the location knowledge released to the SP. The numerous approaches proposed for privacy preservation in location-based services [2] are not directly applicable; in some cases this is because they try to enforce anonymity while users are often explicitly or implicitly identified in social networks, and in other cases because they focus on queries different from the ones needed to compute the proximity service. Section 2 will present in more detail the limitations of these approaches as well as of the recent research efforts that specifically address privacy in proximity-based services. In summary, among the solutions that do take advantage of the SP as a mediator to reduce computation and communications costs, all exhibit one or more of the following problems: i) they do not provide formal guarantees that the information released to the SP, as part of the protocol, cannot be used to violate privacy, ii) they do not consider curious buddies as possible adversaries, and iii) the proposed protocols lead to a significant loss of service precision.

Contribution

Considering this scenario, the main contributions of this paper are the following.

a) This is the first comprehensive rigorous study of location privacy in proximity services, explicitly taking into account privacy control with respect to buddies, providing a way to separate the specification of the distance threshold from the users' privacy requirements.

b) Two new protocols are designed, formally analyzed, and empirically tested, showing their superiority with respect to existing solutions.

The new protocols are proved to guarantee two properties: first, no location information is acquired by the SP, not even in presence of a-priori probabilistic knowledge of users' location; second, for each user, her buddies cannot acquire her location information with a level of precision higher than the one specified by the user. Indeed, each user is allowed to specify her privacy requirements in terms of arbitrary regions of the geographical space: given a specific geographic position of the user, the region including the position defines the highest location precision exposed to her buddies. Differently from other proposals that limit these regions to equally sized cells of a grid, our framework considers arbitrary regions modeled as granules of a spatial granularity.

The first protocol, called *C-Hide&Seek*[5], is shown to provide complete protection with respect to the SP, and to satisfy the privacy requirements of each user with respect to her buddies. Its efficiency is comparable with the simplistic solution adopted in current services for proximity computation that provides no privacy protection. The second protocol, called *C-Hide&Hash*, offers the same guarantees, but provides an even higher level of privacy with respect to the buddies at the cost of higher communication and computation costs. Our solutions are based on the use of well-founded cryptography and secure computation techniques, and require buddies to share a secret. A theoretical analysis of the protocols formally supports the privacy guarantees and evaluates the achieved service precision, as well as the involved computation and communication costs. The practicality of our approach is illustrated by a complete implementation of the techniques in a system, including client applications for mobile phones. Our extensive experimental work reveals the specific behavior of the system in terms of achieved privacy, communication and computation costs, as well as service precision. These statistics indicate that both of our protocols offer significantly better service precision and lower

[5] *C* stands for *centralized*

costs than existing solutions while offering comparable or better privacy. Each of the proposed protocols has its own advantages over the other. *C-Hide&Seek* has very low communication and computation costs leading to high scalability, and it can be used for buddies localization in addition to proximity alerting. *C-Hide&Hash* is a purely proximity alerting protocol, providing a significantly higher privacy than *C-Hide&Seek* at extra communication and computation costs; However, these system costs are still sustainable in practice for reasonably large sets of online buddies, especially when the service is automatically run in background on the mobile device.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we describe more formally the problem we are addressing in terms of privacy concerns, privacy requirements, and adversary models. In Section 4 we illustrate the two proposed protocols, and in Section 5 we study their formal properties, including the satisfaction of privacy requirements, the computational and communication costs, and the service precision. In Section 6 we describe the system implementation, and in Section 7 we report experimental results. Section 8 concludes the paper with a discussion of possible extensions.

2 Related Work

Computing proximity involves the continuous evaluation of spatial range queries over a set of moving entities, with the radius range possibly changing [5,16]. The literature on this problem is both from the database, and the mobile computing community; recent contributions are briefly surveyed in [1], where an efficient algorithm for proximity detection named *Strips* is presented. The goal of this and similar approaches is the efficiency in terms of computation and communication complexity, while privacy issues are mostly ignored.

Proximity services are a special category of location based services (LBS), and several LBS privacy preserving techniques have been recently proposed (see [2] for an extensive survey). So called *anonymity-based* approaches (e.g., [9,10]) consider the possible use of location information contained in anonymous LBS requests coupled with background knowledge to discover the identity of the issuers; in this case, the specific service being requested is often considered private, and privacy is violated when the issuer is identified. Different spatial generalization functions and strategies are proposed to guarantee a given level of anonymity of the users. An anonymized request may contain a quite precise location, since a generalization is considered satisfactory whenever the region contains a sufficiently large number of potential issuers, independently from the actual size of the region. While effective in other scenarios, these techniques are less useful in the proximity services we are considering, since in these services location is considered private information and we don't want to exclude that the identity of buddies may be discovered in other ways. On the contrary, *obfuscation-based* techniques (e.g., [18]) apply transformations on private information, often identified with user location, so that even if the issuer is identified, her location remains protected. These techniques have been applied mostly for LBS performing k-NN spatial queries, and do not apply to proximity detection. Finally, *encryption-based* approaches are inspired to private information retrieval (PIR) methods. The idea of these approaches is to encrypt the information exchanged with the service provider, and to process the corresponding query in an encrypted form, so that no location information is revealed to the SP. The technique proposed in [8] is specifically designed for NN queries, while [11] considers range queries over static resources, which is still not the proper setting for proximity detection, but is indeed a promising approach.

Ruppel et al. [14] propose a technique for privacy preserving proximity computation based on the application of a distance preserving transformation on the location of the users. However, the SP is able to obtain the exact distances between users, and this can lead to a privacy violation. For example, having this knowledge, it is possible to construct a weighted graph of all the users, assigning to each edge connecting two users their exact distance. It is easily seen that a "relative" distribution of the user locations can be extracted from this graph. If the SP has a-priori knowledge about the distribution of the users (as considered in our paper), it is possible to merge the distribution resulting from the graph with the a-priori one, thus revealing some location information about the individuals. In addition, there is no privacy guarantee with respect to the other users participating in the service. The solutions we propose in this paper do not reveal to the SP any information about the distance between users, and let users define the privacy requirement about the location information that buddies can acquire.

Zhong et al. propose three different techniques for privacy preservation in proximity-based services called *Louis*, *Lester* and *Pierre* [19]. These techniques are decentralized secure computation protocols based on public-key cryptography. *Louis* is a three-parties secure computation protocol. By running this protocol, a user A gets to know whether another user B is in proximity without disclosing any other location information to B or to the third party T involved in the protocol. T only

helps A and B compute their proximity, and it is assumed to follow the protocol and not to collude with A or B. However, T learns whether A and B are in proximity. Considering our adversary model, which will be explained in detail in Section 3.3, this third party cannot be the SP that may use proximity information to violate location privacy, and it is unlikely to be played by a third buddy since it would involve significant resources. The *Lester* protocol allows a user A to compute the exact distance from a user B only if the distance between the two users is under a certain threshold chosen by B. The main advantage of these two techniques is that they protect a user's privacy without introducing any approximation in the computation of the proximity. However, *Louis* incurs in significant communication overheads, and *Lester* in high computational costs. In addition, the only form of supported privacy protection with respect to the buddies is the possibility for a user to refuse to participate in the protocol initiated by a buddy if she considers the requested proximity threshold too small. The *Pierre* protocol partitions the plane where the service is provided into a grid, with each cell having edge equal to the requested distance threshold. The locations of the users are then generalized to the corresponding cell, and two users are considered in proximity if they are located in the same cell or in two adjacent cells. The achieved quality of service decreases as the requested proximity threshold grows. We will explain in more detail the actual impact on service precision in Section 7. Finally, it should be observed that Lester and Pierre protocols are based on a buddy-to-buddy communication, and although this can guarantee total privacy with respect to the SP (as no SP is involved in the computation), scalability issues may arise since each time a user moves she needs to communicate her new position to each of her buddies.

Another solution for privacy preserving computation of proximity, called `FriendLocator`, has been proposed by Šikšnys et al. [17]. Similarly to Pierre, two users are considered in proximity when they are located in the same cell or two adjacent cells of the grid constructed considering the proximity threshold shared by the users. An interesting aspect of the proposed solution is the location update strategy, which is designed to reduce the total number of location updates to be sent by the users, hence reducing communication costs. Two users share a hierarchy of grids, where each grid is identified by a *level*. The larger the value of the level is, the finer the grid. The highest level grid is the one in which the edge of a cell is equal to the proximity threshold. The detection of proximity is then incremental, i.e. if two users are in adjacent cells at the level n grid, then their respective cells in the grid of level $n+1$ are checked, until they are detected either not to be in proximity, or to be in proximity considering the highest level grid. With this solution, when two users are detected not to be in proximity at a certain level l, there is no need for them to check again the proximity until one of them moves to a different cell of the level l grid. As a consequence, less location updates are needed, and this is experimentally shown to significantly reduce the total number of messages exchanged. However, the `FriendLocator` protocol reveals some approximate information about the distance of users to the SP (e.g. the level in which the incremental proximity detection protocol terminates and whether the buddies are in proximity at that level). As already observed for the Louis protocol, in our adversary model this information can lead to a privacy violation. Furthermore, the impact on the quality of service of using a large proximity threshold is identical to the Pierre protocol discussed above.

In previous works [12,13], we proposed different protocols for preserving privacy in proximity services. The *Longitude* solution [12] translates the considered space to a toroid, and a distance preserving transformation is applied to the locations of users. The SP participates in a form of three party secure computation of proximity, leading to an approximate but quite accurate service precision, guaranteeing privacy requirements with respect to buddies similar to the ones presented in this paper. Longitude also guarantees complete privacy with respect to the SP under the assumption that he has no a-priori knowledge on the distribution of users, i.e., when a uniform distribution is assumed. In this paper we defend also against SP having arbitrary a-priori distributions, showing that by running our protocols they don't acquire any additional location information. The *Hide&Seek* and *Hide&Crypt* protocols [13] are hybrid techniques in which an initial computation of the proximity condition is done by the SP. In some cases, the SP is not able to decide the proximity condition, and a buddy-to-buddy protocol is triggered. An important difference with respect to the protocols we are presenting here is that the SP is not totally untrusted: users can specify a level of location precision to be released to the SP and (a different one) for buddies. This hybrid approach significantly reduces communication costs with respect to decentralized solutions when privacy requirements with respect to the SP are not too strict.

3 Problem formalization

In this section we formally define the service we are considering, the users' privacy concerns and requirements,

the adversary model, and the occurrence of a privacy violation.

3.1 The proximity service

By issuing a proximity request, user A is interested to know, for each of her buddies B, if the following condition is satisfied:

$$d(loc_A, loc_B) \leq \delta_A \quad (1)$$

where $d(loc_A, loc_B)$ denotes the Euclidean distance between the reported locations of A and B and δ_A is a threshold value given by A. When (1) is true, we say that B *is in the proximity of* A. The proximity relation is not symmetric, since δ_B may be different from δ_A,

In this paper we consider services in which the buddies of a user are pre-determined. We call these services "contact-list-based", since buddies are explicitly added as "friends", like in most social networks and instant messaging applications. This is in contrast to "query-driven" proximity services, in which buddies can be retrieved through a query based, for example, on the interests of the buddies. Technically, the main difference is that in the "contact-list-based" service it is reasonable to assume that each user can share a secret with each of her buddies, as we do in our proposed techniques. On the contrary, in the case of "query-driven" services, the set of buddies may change dynamically, and the number of buddies can be potentially very large. In this situation, it may not be practical to share a secret with each buddy.

With the presence of a service provider (SP), and in absence of privacy concerns, a simple protocol can be devised to implement the proximity service: The SP receives location updates from each user and stores their last known positions, as well as the distance threshold δ_A for each user A. While in theory each user can define different threshold values for different buddies, in this paper, for simplicity, we consider the case in which each user A defines a single value δ_A for detecting the proximity of all of her buddies. When the SP receives a location update, it can recompute the distance between A and each buddy (possibly with some filtering/indexing strategy for efficiency) and communicate the result to A. In a typical scenario, if B is in proximity, A may contact him directly or through the SP; however, for the purpose of this paper, we do not concern ourselves as what A will do once notified. In the following of this paper we refer to the above protocol as the *Naive* protocol.

3.2 Privacy concerns and privacy requirements

The privacy we are considering in this paper is *location privacy*: we assume that a user is concerned about the uncontrolled disclosure of her location information at specific times.

Considering the *Naive* protocol, it is easily seen that the SP obtains the exact location of a user each time she issues a location update. Furthermore, a user's location information is also disclosed to her buddies. If Alice is in the proximity of Bob (one of her buddies), then Bob discovers that Alice is located in the circle centered in his location with radius δ_{Bob}. Since δ_{Bob} is chosen by Bob and can be set arbitrarily without consent from Alice, Alice has no control on the location information disclosed to Bob.

Our definition of location privacy is based on the idea that the users should be able to control the location information to be disclosed. In the considered services, a user may prefer the service provider to have as little information about her location as possible, and the buddies not to know her exact position, even when the proximity is known to them. Moreover, the exchanged information should be protected from any eavesdropper.

In general, the level of location privacy can be represented by the uncertainty that an external entity has about the position of the user. This uncertainty is a geographic region, called *minimal uncertainty region* (MUR), and its intuitive semantics is the following: the user accepts that the adversary knows she is located in a MUR R, but no information should be disclosed about her position within R.

In the solution proposed in this paper, each user can express her privacy preferences by specifying a partition of the geographical space defining the MURs that she wants guaranteed. For example, Alice specifies that her buddies should never be able to find out the specific campus building where Alice currently is; in this case, the entire campus area is the minimal uncertainty region. The totality of these uncertainty regions for a user can be formally captured with the notion of *spatial granularity*.

While there does not exist a formal definition of spatial granularity that is widely accepted by the research community, the idea behind this concept is simple. Similar to a temporal granularity [3], a spatial granularity can be considered a subdivision of the spatial domain into a discrete number of non-overlapping regions, called *granules*. In this paper, for simplicity, we consider only granularities[6] that partition the spatial

[6] Here and in the following, when no confusion arises, we use the term "granularity" to mean "spatial granularity".

domain, i.e., the granules of a granularity do not intersect and the union of all the granules in a granularity yields exactly the whole spatial domain. Each granule of a granularity G is identified by an *index* (or a *label*). We denote with $G(i)$ the granule of the granularity G with index i.

Users specify their *privacy requirements* via spatial granularities, with each granule being a MUR. The two extreme cases in which a user requires no privacy protection and maximum privacy protection, respectively, can be naturally modeled. In one extreme case, if a user A does not want her privacy to be protected then A sets her privacy preference to the *bottom granularity* $\perp$ (a granularity that contains a granule for each basic element, or pixel, of the spatial domain). In the other extreme, if user A wants *complete* location privacy then she sets her privacy preference to the *top granularity* $\top$, i.e., the granularity that has a single granule covering the entire spatial domain. In this case, A wants the entire spatial domain as MUR.

In the following of this paper, we assume that each user A specifies a granularity G_A defining her location privacy requirements with respect to all buddies. Our approach can be easily extended to model the case in which a user specifies a different granularity for a different buddy or for a different group of buddies, as discussed in Section 8. We also assume that each user's privacy requirement with respect to the SP is the entire spatial domain, i.e., the user does not want to disclose any location information to the SP.

3.3 Adversary model and privacy preservation

We consider two adversary models, for the SP and the buddies, respectively. Assuming the SP and the buddies as potential adversaries, also models other types of adversaries. Firstly, it models the case of an external entity taking control of the SP system or of a buddy's system. Secondly, it models the case of an external entity eavesdropping one or more communication channels between users and the SP. Note that, in the worst case, the eavesdropper can observe all the messages that are exchanged in the protocol. Since the same holds for the SP, the eavesdropper can learn at most what the SP learns. Since in this paper we prove that the SP does not acquire any location information, then the same holds for an eavesdropping adversary.

The techniques we present in this paper not only guarantee each user's privacy requirement against these two adversary models, but also in the case of a set of colluding buddies. In Section 5.1.2 we also discuss which privacy guarantees are provided by our techniques in case one or more buddies collude with the SP.

In both adversary models we assume that the adversary knows:

- the protocol,
- the spatial granularities adopted by each user, and
- an a-priori probabilistic distribution of the locations of the users.

The two models differ in the sets of messages received during a protocol run, and in their ability (defined by the protocol in terms of availability of cryptographic keys) to decrypt the content of the messages.

The a-priori knowledge of the location of a user A is given by a location random variable pri_A with the probability mass distribution denoted $P(pri_A)$. In other words, as prior knowledge we assume that the location of a user A follows a known distribution given by the distribution of the random variable pri_A. Note that in this paper we assume the spatial domain is discrete, i.e., a countable set of "pixels".

Let M be the set of messages exchanged between the entities involved in the service. The adversary can compute the *a-posteriori* probability distribution of the location random variable $post_A$ as the distribution of the location of A under the given messages M and the prior knowledge pri_A:

$$P(post_A) = P(loc_A|M, pri_A)$$

Technically, we may view loc_A as a uniform random variable over the spatial domain, i.e., the possible location of A when no knowledge is available.

The condition for privacy preservation is formally captured by Definition 1.

Definition 1 Given a user A with privacy requirement G_A, and M the set of messages exchanged by the proximity service protocol in which A is participating, A's privacy requirement is said to be *satisfied* if

$$P(loc_A|M, pri_A, loc_A \in g_A) = P(loc_A|pri_A, loc_A \in g_A)$$

for all a-priori knowledge pri_A and all granule g_A of G_A.

The above definition requires that the location distribution of user A does not change due to the messages M, given the a-priori knowledge and the fact that A is located in g_A. Hence, a privacy violation occurs when the adversary acquires, through the analysis of the protocol messages, *more* information about the location of A than allowed by her privacy requirements, i.e., when the probability distribution of the position of A within the region defined by granule g_A changes with respect to pri_A.

Since we aim at complete location privacy with respect to the SP, we use g_A to be the entire spatial domain in the above definition when the SP is concerned.

In this case, the definition requires $P(loc_A|M, pri_A) = P(loc_A|pri_A)$, i.e., $P(post_A) = P(pri_A)$ or *no new location information* for each user A. In this case, we also say that A's privacy requirement is *satisfied with respect to the SP*. For the buddies, user A uses a granularity G_A, which may not be $\top$. In this case, the definition requires that with the additional knowledge of A being in a granule, the buddies cannot derive anything more (e.g., where within the granule) from the messages exchanged. In this case, we also say that A's privacy requirement is *satisfied with respect to the buddies*.

4 Defense techniques

In this section we present two protocols to preserve location privacy in proximity-based services. The protocols are called *C-Hide&Seek* and *C-Hide&Hash* and they guarantee privacy protection of a user A with respect to both the SP and the buddies of A.

In order to ensure user's privacy, the two protocols adopt symmetric encryption techniques. In the following, we assume that each user A has a key K_A that is shared with all of her buddies and is kept secret to everybody else. Hence, each user A knows her own key K_A and one key K_B for each buddy B. Since we are considering a contact-list-based service, this key exchange is assumed to be performed with any secure method before running our protocols.

For the sake of presentation, we decompose each protocol into two parts: the *location update* sub-protocol is used by a user to provide her location information, while the *proximity request* sub-protocol is used by a user to compute the proximity of her buddies. The location update sub-protocol is almost the same in both of our proposed solutions, and it is presented in Section 4.1. What really distinguishes *C-Hide&Seek* and *C-Hide&Hash* is the proximity request sub-protocol, and this is described in Sections 4.2 and 4.3, respectively. We conclude this section with a discussion about possible technical extensions.

4.1 The location update sub-protocol

The location update sub-protocol is run by a user to provide location information to the SP. In particular, it defines how a user A provides to the SP the encrypted index of the granule of G_A where she is located.

Before describing the sub-protocol, we first discuss when it should be run. Consider the following naive policy: a user A updates her location only when she crosses the boundary between two granules of G_A, reporting the index of the new granule. It is easily seen that, independently from how the location update is performed, each time this message is received, the adversary learns that A is very close to the border between two granules, excluding many other locations, and hence violating the privacy requirements. Intuitively, the problem of the above policy is that the probability that a location update is performed at a given time depends on the location from where the message is sent.

The solution we propose is the following: time is partitioned into *update intervals* and an approximate synchronization on these intervals among the participating nodes is assumed.[7] Each update interval has the same duration T and is identified by an index. Each user has a value t in $[0, T)$ and sends exactly one location update during each update interval after that time t elapses from the beginning of the interval (see Figure 1). It is easily seen that, by using this update policy, the location updates are issued independently from the location of the users.

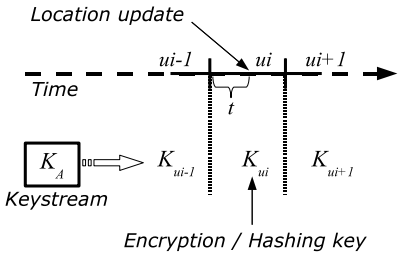


Fig. 1 Location update policy and generation of single-use keys.

We now describe how the location update sub-protocol works. User A first computes the index i of the granule of G_A where she is located. Then, A encrypts i using a slightly different technique in the two proposed solutions. In the *C-Hide&Seek* protocol a symmetric encryption function E is applied, while in the *C-Hide&Hash* protocol a hashing function H is used. When applying the hashing function H, in order to prevent brute-force attacks, a secret key is used as a "salt", i.e., a secret key is concatenated to i, and the resulting value is given as input to H. In the following, we refer to this salt as the "key" used to hash i, and we denote with $H_K(i)$ the hashing of the value i with key K.

The safety of the protocols depends on the fact that the key used to encrypt or hash i is changed at every use. At the same time, we need the key to be shared by a

[7] In our current implementation, all the messages sent from the SP to the users contain the timestamp of the SP, allowing clients to synchronize their clocks using a Lamport-style algorithm. The overhead due to this solution is negligible. Other forms of global clock synchronization could also be used as, e.g., using GPS devices.

user with all of her buddies. While other techniques can be adopted to achieve this result, our solution is the following: the key K_A that A shares with all of her buddies is used to initialize a keystream. When user A issues a location update, she computes the key K^{ui} as the ui-th value of this keystream, where ui is the index of the current update interval (see Figure 1). Since each user issues a single location update during each time interval, this solution ensures that every message is encrypted or hashed with a different key. Finally, A sends to the SP the message $\langle A, ui, E_{K^{ui}}(i)\rangle$ if running *C-Hide&Seek*, and $\langle A, ui, H_{K^{ui}}(i)\rangle$ if running *C-Hide&Hash*. The SP stores this information as the last known encrypted location for A. Figure 2 shows the message sent from A to the SP by the *C-Hide&Seek* protocol.

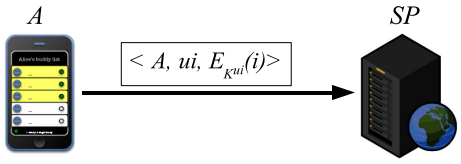


Fig. 2 Location update sub-protocol in *C-Hide&Seek*.

4.2 Proximity request with *C-Hide&Seek*

The proximity request sub-protocol is run by a user that wants to discover which of her buddies are in proximity. In the *C-Hide&Seek* protocol, this sub-protocol works as follows: When A wants to discover which buddies are in proximity, she sends a request to the SP. The SP replies with a message containing the last known encrypted location of each buddy of A. That is, for each buddy B, A receives a tuple $\langle B, ui, E_{K^{ui}}(i)\rangle$. Since A knows K_B and the index ui is in the message, she can compute the value K^{ui} used by B to encrypt his location, and hence she can decrypt $E_{K^{ui}}(i)$. Finally, since A also knows G_B, by using i, she obtains the granule $g_B = G_B(i)$ where B is located. A can then compute the distance between her exact location and g_B, and compare it with δ_A, finally determining the proximity. Figure 3 shows a graphical representation of the sub-protocol.

Note that we are now considering the proximity between a point and a region. In this section, we consider that a point and a region are in proximity, with respect to a distance threshold, if the *minimum* distance between the two objects is less than the threshold. Since, in our protocol, the region represents the area where a user B is possibly located, this interpretation of proximity means that there is a possibility for users A and B to

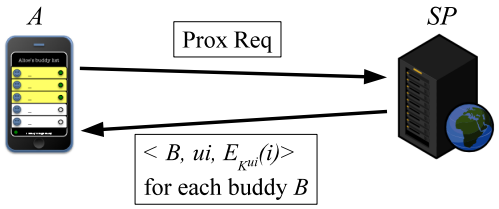


Fig. 3 Proximity request sub-protocol in *C-Hide&Seek*.

actually be in proximity. The same *minimum* distance interpretation has been used in related work on privacy-aware proximity computation. Alternative interpretations and their effects are discussed in Section 5.2.

The *C-Hide&Seek* protocol provides a simple and efficient solution that, as will be shown in Section 5, completely hides the location of the users to the SP, and that also guarantees the privacy requirements with respect to the buddies. However, it reveals exactly the maximum tolerable amount of location information (g_B for user B) to any buddy issuing a proximity request. Even if their privacy requirements are guaranteed, users would probably prefer to disclose as little information as possible about their location when not strictly needed. For example, is there an alternative solution that does not reveal to a user A the granule information of a buddy B if he is not in proximity?

In the next section we present the *C-Hide&Hash* protocol that provide such a solution and, in general, ensures a higher level of privacy. This is achieved at the cost of higher computation and communication costs, as explained in Section 5.4.

4.3 Proximity request in *C-Hide&Hash*

The *C-Hide&Hash* protocol has two main differences with respect to *C-Hide&Seek*. The first difference is that a hash function H is used during the location update, instead of the encryption function. This is due to the requirement in this protocol to avoid revealing the relationship between two plaintext values (the granule indexes) by observing the relationship among the corresponding encrypted values (see Section 5 for a more detailed explanation). Since in this protocol we do not need to decrypt the result of the function, but we only need to check for equality of encrypted values, hashing can be used. As specified in Section 4.1, each location update in *C-Hide&Hash* from user A to the SP is a message containing the tuple $\langle A, ui, H_{K^{ui}}(i)\rangle$.

The second and main difference with respect to *C-Hide&Seek* is the computation of the proximity request sub-protocol. The intuition is that when A issues a proximity request, she computes, for each of her buddies B, the set of indexes of granules of G_B such that, if

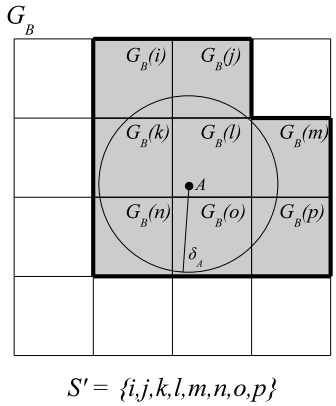


Fig. 4 Computation of granules of G_B considered in proximity by A

B is located in any granule of the set, then B is in proximity (see Figure 4). Then, if B provides the granule in which he is located, it is possible to reduce the proximity problem to the set-inclusion problem, by checking if that granule is included in the set computed by A. We want to do this set inclusion without revealing to A which of the candidate granules actually matched the granule of B.

More precisely, the computation of a proximity request in the *C-Hide&Hash* protocol works as follows. When a user A issues a proximity request, she starts a two-party set inclusion protocol with the SP. The protocol is a secure computation, and consequently the SP does not learn whether A is in proximity with her buddies, and A only learns, for each of her buddies B, whether B is in proximity or not, without learning in which granule B is located. The secure computation exploits a commutative encryption function C. In addition to the keys used in the *C-Hide&Seek* protocol, at each proximity request, the requesting user and the SP each generates a random key that is not shared with anyone else. We denote these keys K_1 for user A and K_2 for the SP.

The proximity request sub-protocol is divided into three steps, whose pseudo-code is illustrated in Protocol 1. In Step (i), user A computes, for each buddy B, the set S' of indexes of granules of G_B such that, if B is located in one of these granules, then B is in proximity. More formally, A computes the set of indexes i such that the minimum distance *minDist* between the location of A and $G_B(i)$ is less than or equal to δ_A. Then, in order to hide the cardinality of S', A creates a new set S by adding to S' some non-valid randomly chosen indexes (e.g., negative numbers). This is done to increase the cardinality of S without affecting the result of the computation. The cardinality of S is increased so that it is as large as the number $sMax(G_B, \delta_A)$ that represents the maximum number of granules of G_B that intersect with any circle with radius δ_A. Note that $sMax(G_B, \delta_A)$ can be computed off-line since its values depend only on G_B and δ_A. In the following, when no confusion arises, we use $sMax$ as a short notation for $sMax(G_B, \delta_A)$.

Protocol 1 *C-Hide&Hash*: proximity request

Input: User A knows, the last completed update interval, and the proximity threshold δ_A. Also, for each of her buddy B, A knows the granularity G_B, the key K_B and the value of $sMax(G_B, \delta_A)$.
Protocol:
(i) Client request from A
1: $proxReq = \emptyset$
2: generate a random key K_1
3: **for** each buddy B of A **do**
4: $S' = \{j \in \mathbb{N}$ s.t. $minDist(loc_A, G_B(j)) \leq \delta_A\}$
5: S'' = a set of $sMax(G_B, \delta_A) - |S'|$ non-valid random indexes.
6: $S = S' \cup S''$
7: K^{ui} is the ui-th value of the keystream initialized with K_B
8: $ES = \bigcup_{i \in S} C_{K_1}(H_{K^{ui}}(i))$
9: insert $\langle B, ui, ES \rangle$ in $proxReq$
10: **end for**
11: A sends $proxReq$ to the SP
(ii) SP response
1: $proxResp = \emptyset$
2: generate a random key K_2
3: **for** each $\langle B, ui, ES \rangle$ in $proxReq$ **do**
4: $ES' = \bigcup_{e \in ES} C_{K_2}(e)$
5: retrieve $\langle B, ui, h_B \rangle$ updated by B at update interval ui
6: $h' = C_{K_2}(h_B)$
7: insert $\langle B, ES', h' \rangle$ in $proxResp$
8: **end for**
9: SP sends $proxResp$ to A
(iii) Client result computation
1: **for** each $\langle B, ES', h' \rangle$ in $proxResp$ **do**
2: $h'' = C_{K_1}(h')$
3: **if** $h'' \in ES'$ **then**
4: A returns "B is in proximity"
5: **else**
6: A returns "B is not in proximity"
7: **end if**
8: **end for**

In Line 8, each element of S is first hashed using the key K^{ui}, which is obtained as the ui-th value generated by the keystream initialized with K_B. In this case ui is the index of the update interval preceding the current one. Then, the result is encrypted, using the commutative encryption function C and key K_1 that is randomly generated. The element composed by the set ES computed in Line 8, B, and ui is then added to the set *proxReq*.

Once the operations in Lines 4 to 9 are executed for each buddy B, the set *proxReq* is sent to the SP.

Upon receiving *proxReq*, the SP starts Step (ii). For each tuple $\langle B, ui, ES \rangle$ in *proxReq*, the SP encrypts with the C function each element of ES using key K_2, which

is randomly generated. The result is the set ES'. Then, it retrieves the tuple $\langle B, ui, h_B \rangle$ updated by B at the update interval ui. In this tuple, h_B is the value of the index of the granule of G_B where B is located, hashed with the key K^{ui}. Since ui is the update interval preceding the current one, our location update policy assures that a location update with update interval ui has already been issued by every buddy B. Finally, the SP encrypts h_B with the commutative encryption function C using key K_2. The resulting value h' is added, together with B and ES', to the set *proxResp*.

Once the computations at Lines 4 to 7 are executed for each buddy B, the set *proxResp* is sent to A.

In Step (iii), given the message *proxResp* received from the SP, A computes the proximity of her buddies. For each tuple $\langle B, ES', h' \rangle$, A obtains h'' as the encryption of h' with C and the key K_1 and checks if the result is in ES'. If this is the case, then B is in proximity, otherwise he is not.

More formally, $h'' \in ES'$ if and only if the granule of G_B with index i containing B is in S', that is equivalent to B being in proximity. Indeed, for each buddy B, we recall that:

$$h'' = C_{K_1}(C_{K_2}(h_B))$$

and

$$ES' = \bigcup_{i \in S} (C_{K_2}(C_{K_1}(H_{K^{ui}}(i)))$$

Consequently, due to the commutative property of the encryption function, $h'' \in ES'$ if and only if

$$h_B \in \bigcup_{i \in S} H_{K^{ui}}(i)$$

Since h_B and the elements of the set are hashed using the same key K^{ui}, h_B is in the set if and only if $i \in S$. Since $S = S' \cup S''$ and $i \notin S''$ (because S'' contains invalid integers only while i is a valid integer) then $i \in S$ if and only if $i \in S'$. By definition of S', this implies that B is in proximity.

Figure 5 shows the messages exchanged during the proximity request sub-protocol of *C-Hide&Hash*.

4.4 Contrasting velocity attacks and other background knowledge

It is easily seen that our location update policy, based on fixed length update intervals, makes the probability that a location update is issued independent from the location from where it is issued. This is an important property used in Section 5, together with others, to prove the safety of our solutions under the adversary models we consider.

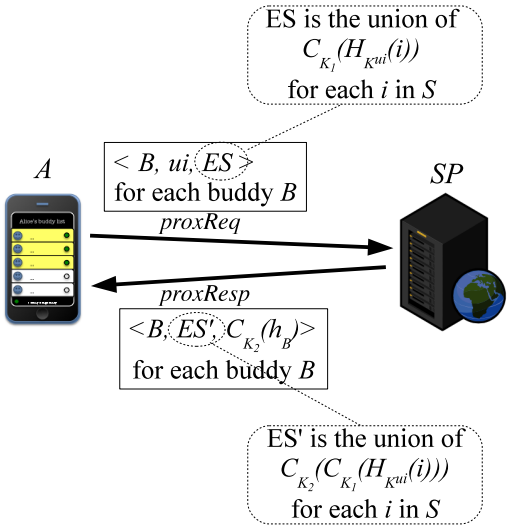


Fig. 5 Proximity request sub-protocol in *C-Hide&Hash*.

Clearly, if the adversary had arbitrary background knowledge, there would not be any technique that could guarantee privacy. However, it is interesting to consider some other forms of knowledge that the adversary could use. With respect to previous proposals, our defenses are resistant to an important type of background knowledge: a-priori distribution of the users' locations. There are, however, other types of knowledge that may be interesting to consider as, for example, the *time-dependent* a-priori location knowledge. This includes knowledge on the *relative* position of users at a certain time, as well as a-priori probability of user movements. With this kind of knowledge it is also possible to perform attacks based on the velocity of users. Consider Example 1.

Example 1 User A sends two location updates in two consecutive update intervals i and j from granule g_1 and g_2, respectively. Her buddy B issues a proximity request in each update interval and discovers the granule where A is located. So far, no privacy violation occurred for A. However, if B knows that A moves at most with velocity v, then he can exclude that A is located in some locations l of g_2. Indeed, B knows that the temporal distance between the two location updates of A is equal to the length T of the update period. Now B can exclude that A is located in any location l of g_2 such that the time required to move from any point of g_1 to l with velocity v is larger than T. Hence B violates the privacy requirement of A.

The problem in Example 1 arises when the adversary knows the maximum velocity of a user. Velocity-based attacks have been recently considered independently from proximity services [7], but the application

of those solutions in our framework would lead to the release of some location information to the SP. In the following we show how to adapt our location update policy to provide protection preserving our privacy properties in the specific case in which the adversary knows the maximum velocity v of a user.

Let $tMax(g_1, g_2)$ be the maximum time required to move at velocity v from each point of granule g_1 to each point of granule g_2. The problem of Example 1 arises when the temporal distance between two location updates issued from two different granules g_1 and g_2 is less then $tMax(g_1, g_2)$. The problem can be solved by imposing that A, after entering g_2, randomly reports g_1 or g_2 as the granule where she is located until time $tMax(g_1, g_2)$ elapses from the last location update in g_1. This solution is a form of temporal generalization as it adds uncertainty to the adversary, about when the user crosses the border between g_1 and g_2. More specifically, the adversary is unable to identify the exact instant in which the user crossed the border in a time interval of length at least $tMax(g_1, g_2)$. Consequently, by definition of $tMax(g_1, g_2)$, the adversary cannot exclude that A moved from any point of g_1 to any point of g_2.

The extension of our defense techniques to other forms of background knowledge is one of the subjects for future work.

5 Analysis of the protocols

The main goal of our techniques is to guarantee the satisfaction of users' privacy requirements under the given adversary models. In Section 5.1 we prove that our two protocols have this property.

However, there are other important parameters to be considered in an evaluation and comparison among protocols that satisfy the privacy requirements. In general, the higher the privacy provided by the protocol, the better is for the users; since location privacy in our model is captured by the *size of the uncertainty region*, in Section 5.3 we consider this parameter.

A second parameter to be considered is *service precision*. The percentage of false positives and false negatives introduced by a specific protocol must be evaluated. This is considered in Subsection 5.2.

Last but not least, it is important to evaluate the overall *system cost*, including computation and communication, with a particular attention to client-side costs. This is considered in Subsection 5.4.

The proofs of the formal results presented in this section are in Appendix A.

5.1 Privacy

We first analyze the privacy provided by *C-Hide&Seek* and *C-Hide&Hash* in Section 5.1.1 considering the adversary models presented in Section 3 under the *no-collusion* assumption, i.e., assuming that the SP does not collude with the buddies and that the buddies do not collude among themselves. Then, in Section 5.1.2 we show the privacy guarantees provided by the two algorithms in the more general case of possibly colluding adversaries.

5.1.1 Satisfaction of privacy requirements

We first analyze the *C-Hide&Seek* protocol. Since the private key K_A is only known to A and to the buddies of A, the SP is not able to decrypt the index of the granule where A is located. Analogously, the SP is not able to obtain location information about A's buddies and, in particular, does not obtain any information about the distance between A and her buddies.

We now state a formal property of the *C-Hide&Seek* that is used in the formal proof of the above observations.

Lemma 1 *The C-Hide&Seek protocol ensures that under any a-priori knowledge pri_A, the following two random variables are probabilistically independent: (1) The binary random variable $ur(A)$: an update/request is sent by user A, and (2) random variable loc_A, i.e., the location of A, of any distribution. Formally, we have*

$$P(ur(A)|loc_A, pri_A) = P(ur(A)|pri_A),$$

for any a-priori location knowledge pri_A and location random variable loc_A for user A.

Note that we are assuming discrete time and discrete location. A continuous case can be formalized and proved equally easily. Also, this lemma does not concern the type or content of a message sent by A, but just the fact that a message is sent by A.

Another property we use to prove our safety result is provided by the encryption algorithms, via the information theoretical notion of "perfect secrecy" [4]. Intuitively, perfect secrecy for an encryption algorithm means that given ciphertext c, each plaintext p has the same probability to be encrypted to c (posterior), with a randomly chosen key, as the probability of p to be used in the first place (prior). That is, $P(p|c) = P(p)$. Equivalently, given plaintext p, each ciphertext c has the same probability to be the encryption of p (posterior), with a randomly chosen key, as the probability of c to appear in the first place as ciphertext (prior). That is, $P(c|p) = P(c)$. Applied to our situation, when SP

receives a message $\langle A, ui, E_{K^{ui}}(l)\rangle$, since K^{ui} is hidden from the SP and can be chosen arbitrarily, the probability that SP receives any other message of the form $\langle A, ui, E_{K^{ui}}(l')\rangle$ is the same.

Most of practical encryption algorithms do not have the theoretical perfect secrecy, but use computational hardness to achieve secrecy in the sense that it is computationally very hard (or impractical) to derive the plaintext from the ciphertext. Intuitively, $P(p|c) = P(p)$ holds because c does not yield any information about p. Therefore, we use the simplifying, practical assumption that the encryption methods we use do give us perfect secrecy.

The above perfect secrecy discussion applies to single messages. When dealing with multiple messages, correlation between plaintexts may reveal secrets when the same key is used. This is the classical scenario of repeated key use problem, and one solution to this problem is to use so-called one-use-pad or keystreams as we do in our proposed protocols. As each key is only used once, encrypted messages are independent to each other when perfect secrecy is assumed.

From the above discussion and assumptions, Lemma 2 follows. Since the lemma involves random variables on messages, we need to specify the *message space* for these variables. We consider the randomness of the messages to be on the encrypted part, while other parts are fixed. Formally, we call each sequence $\langle B_1, ui_1\rangle, \ldots, \langle B_n, ui_n\rangle$, where B_j is a user and ui_j is a time interval, a *(message set) type*. (Recall that a message is of the form $\langle B, ui, ES\rangle$.) The messages of the same type differ on the encrypted part of the messages and constitute a message space. When a generic message M is mentioned, we assume it is a variable over all the messages with a specific type.

Lemma 2 *Given messages $M = M_1 \cup M_2$ issued in the C-Hide&Seek protocol, where $M_1 \cap M_2 = \emptyset$, we have*

$$P(M|loc_A, pri_A) = P(M_1|loc_A, pri_A) * P(M_2|loc_A, pri_A),$$

for all a-priori knowledge pri_A and location loc_A for user A.

With Lemma 1, perfect secrecy, and Lemma 2, we now show a main result, namely, the SP does not acquire any location information as a consequence of a location update or a proximity request using the *C-Hide&Seek* protocol. The following formal results implicitly refer to our adversary models that, in particular, assume that the SP has no background knowledge other than the protocol, the a-priori distribution, and the granularities.

Theorem 1 *Let A be a user issuing a sequence of location updates and proximity requests following the C-Hide&Seek protocol. Then, A's privacy requirement is satisfied with respect to the SP.*

We now turn to the location information acquired by the buddies. In the *C-Hide&Seek* protocol, a user A issuing a proximity request does not send any location information, hence her buddies, even if malicious, cannot violate her privacy requirements. When the same user runs the location update subprotocol in *C-Hide&Seek*, her buddies can only obtain the granule at the granularity G_A in which A is located. As a consequence, the privacy requirement of A is guaranteed. This is formally stated in Theorem 2.

Theorem 2 *Let A be a user issuing a sequence of location updates and proximity requests following the C-Hide&Seek protocol. Then, A's privacy requirement is satisfied with respect to each of A's buddies.*

We consider now the *C-Hide&Hash* protocol. Since K_A is only known to A and her buddies, the SP is not able to acquire the location information provided by A during a location update. This follows from Theorem 1. The difference of the *C-Hide&Hash* from the *C-Hide&Seek* is that when A issues a proximity request in *C-Hide&Hash*, an encrypted message is sent to the SP. However, due to the property of the secure computation protocol in *C-Hide&Hash*, the only information that the SP acquires about the set provided by A is its cardinality. Actually, the cardinality of this set is always S_{MAX} that, by definition, depends only on δ_A and G_B, and not on the actual location of A or B. Consequently, the SP does not acquire any information about the location of A and B, including their distance. Theorem 3 formally states this property.

Theorem 3 *Let A be a user issuing a sequence of location updates and proximity requests following the C-Hide&Hash protocol. Then A's privacy requirement is satisfied with respect to the SP.*

Similarly to the *C-Hide&Seek* protocol, in *C-Hide&Hash* each buddy of A can only obtain location information derived from A's location update. It is worth noting that in the *C-Hide&Seek* protocol, each time B issues a proximity request, he obtains the granule of G_A where his buddy A is located. Differently, using the *C-Hide&Hash* protocol, B only gets to know whether the granule where A is located is one of those in S_A. This means that, if A is not in proximity, then B only learns that A is not in any of the granules of S_A. Otherwise, if A is in proximity, B learns that A is in one of the granules of S_A, without knowing exactly in which granule she is located. This is formally stated in Theorem 4.

Theorem 4 *Let A be a user issuing a sequence of location updates and proximity requests following the C-Hide&Hash protocol. Then, A's privacy requirement is satisfied with respect to each of A's buddies.*

In Section 7 we show that, on average, *C-Hide&Hash* provides more privacy with respect to the buddies than *C-Hide&Seek*, but at extra costs, making each protocol more adequate than the other based on user preferences and deployment modalities.

5.1.2 Privacy in case of possibly colluding adversaries

We now consider the case in which our reference adversaries can collude, and we analyze the privacy guarantees of the *C-Hide&Hash* and *C-Hide&Seek* protocols in this scenario.

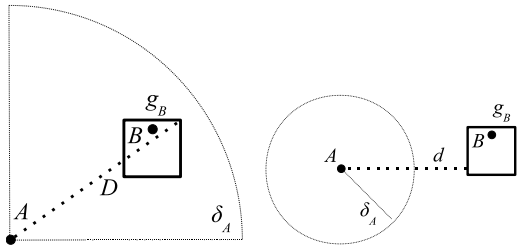
First, consider the case in which two buddies B and C collude to violate the privacy of a user A. The problem can be easily extended to consider more buddies. Let l_B be the set of possible locations of A obtained by B as a result of a proximity request. Let l_C be the analogous information acquired by C during the same update interval. Since B and C collude, they can derive that A is located in $l_B \cap l_C$. However, due to Theorem 4, given $G_A(i)$ the granule where A is located, it holds that $l_B \supseteq G_A(i)$ and $l_C \supseteq G_A(i)$ (recall that G_A is the privacy requirement of A with respect to the buddies). Consequently, $l_B \cap l_C \supseteq G_A(i)$ and hence the privacy requirement of A is guaranteed also in the case B and C collude.

Now, consider the case in which the SP colludes with one or more buddies. For example, if one of the buddies shares the secret key K_A with the SP, the SP can learn the granule where A is located. In this case, the privacy requirement of A with respect to the SP is not guaranteed. Nevertheless, even if the SP knows K_A, he cannot discover the location of A within the granule of $G_A(i)$ where A is located. This is because, by the definition of the two protocols, every message issued by A does not depend on the location of A within $G_A(i)$. Consequently, the privacy requirement with respect to the buddies is still guaranteed. This means that the lowest privacy requirement of the two colluding entities is preserved and this is the best that can be achieved in case of collusion.

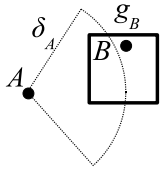
5.2 Service precision

The techniques proposed in the literature as well as the techniques we propose in this paper, generalize the location of one of the two users to an area. When proximity is computed, the exact location of that user within the area is not known. Hence, proximity is evaluated as the distance between a point and a region[8].

Consider how it is possible to compute the proximity between a user A whose exact location is known and a user B whose location is only known to be in region. It is easily seen that if the maximum distance between the point and the region is less than the proximity threshold, then the two users are in proximity, independently from where B is located within the region. Figure 6(a) shows an example of this situation. On the contrary, if the minimum distance is larger than the distance threshold, then the two users are not in proximity. Figure 6(b) graphically shows that this happens when no point of the region containing B is in proximity of A. If none of the two cases above happen (i.e., the threshold distance is larger than the minimum distance and less than the maximum distance), we are in presence of an *uncertainty case*, in which it is not possible to compute whether the two users are in proximity without introducing some approximation in the result. For example, Figure 6(c) shows that if B is located close to the bottom left corner of the region then B is in the proximity of A, otherwise he is not.



(a) B is in proximity of A (b) B is not in proximity of A



(c) B is *possibly* in proximity of A

Fig. 6 Regions L_A and L_B

The choice we made in the presentation of our protocols is to consider two users as in proximity in the *uncertainty case*. The rational is that in this case it is not possible to exclude that the users are not in proximity. Previous approaches ([19,13]) facing a similar issue have adopted the same semantics.

[8] In previous work, the location of both users is generalized and proximity is computed between two regions.

One drawback of this *minimum-distance* semantics is that it generates false positive results and this may be undesirable in some applications. Indeed, if user B is reported to be in proximity of A, then A may decide to contact B (e.g., through IM). This may be annoying for B, if he is not actually in proximity. Consider, for example, the case in which the location of B is reported at the granularity of a city: B is always reported as in proximity of A when A is in the same city, independently from the proximity threshold chosen by A.

An alternative semantics, that we name *maximum-distance* semantics, solves this problem. The idea is to consider two users as in proximity only when it is certain that they are actually in proximity. This happens when the maximum distance between their areas is less than the distance threshold. While this approach does not generate any false-positive, it does produce false-negatives. The two semantics above have a common drawback: in certain cases it happens that the probability of providing a false result is larger than the probability of providing a correct result. Consider the example depicted in Figure 7 in which the *minimum-distance* semantics is considered. User B is considered in proximity but the answer is wrong if B is located in the region colored in gray. Assuming a uniform distribution of B inside g_B, it is much more likely to have an incorrect result, rather than a correct one. An analogous problem can arise for the *maximum-distance* approach.

The percentage of false results can be minimized by considering user B as in proximity only when at least one half of the area is actually in proximity. The drawback of this *mostly-in-proximity* semantics is that it incurs in both false positive and false negative results.

Our protocols are designed so that it is very easy to change the current proximity semantics. Since this can be done client-side, without the need for changes server-side nor in the code other peers are running, the semantics can be potentially chosen through the user interface at any time.

We analytically measured the impact of the different semantics on the accuracy of our protocols by calculating the *expected precision* and the *expected recall*. The expected precision is defined as the probability that a buddy reported to be in proximity according to a given semantic is actually in proximity. Vice versa, the expected recall is defined as the probability that a buddy actually in proximity is reported to be in proximity according to a given semantic.

Figures 8 and 9 show the minimum expected precision and recall for the *minimum-distance* and the *maximum-distance* semantics. Both measures depend on the ratio between δ and the area of the granules in which a user is considered in proximity. For this analysis we considered a grid-like granularity containing cells having edge of size l and we assume users are uniformly distributed. As can be observed in Figure 8, the *maximum-distance* semantic has always precision equal to 1. This is because all the buddies considered in proximity are always actually in proximity. The *minimum-distance* has precision of about 1/3 when the values of δ and l are equal, and this value grows logarithmically when δ is larger than l. The analysis of expected recall (Figure 9) shows that the *minimum-distance* has

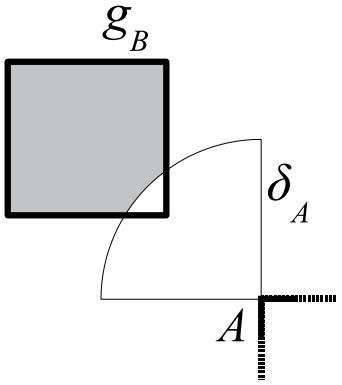


Fig. 7 Approximation incurring with the *minimum-distance* semantics

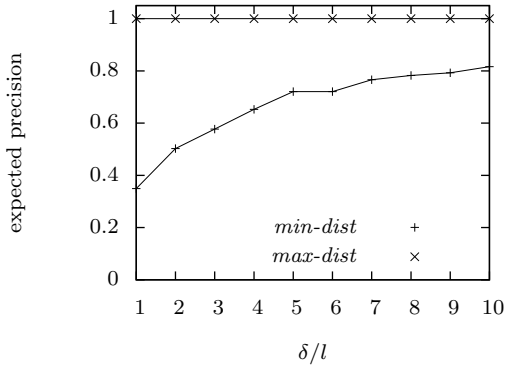


Fig. 8 Expected precision

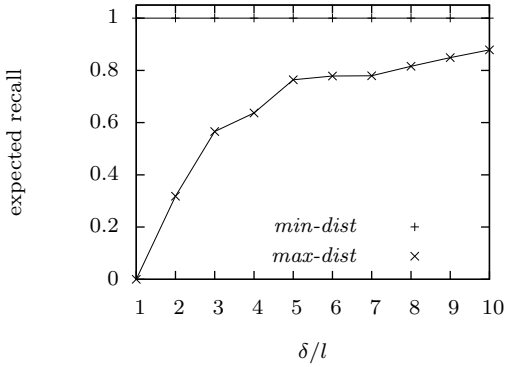


Fig. 9 Expected recall

always recall equal to 1. This is because if a buddy is actually in proximity, it is always reported in proximity using this semantic. The *maximum-distance* semantic, on the contrary, has a minimum expected recall equal to 0 when δ and l are equal. This is because, with this parameters, it can happen that no cells of size l are fully contained in a circle having radius δ. However, the recall of the *maximum-distance* grows more rapidly than the precision of the *minimum-distance*.

5.3 Size of uncertainty regions

As already discussed in Section 5.1, our protocols are proven to always guarantee the privacy requirement with respect to the buddies. However, the main difference between our two protocols consists in the fact that *C-Hide&Hash* can provide additional privacy with respect to one buddy. For example, if a user A issues a proximity request using *C-Hide&Hash*, and a buddy B is reported as being not in proximity, A only learns that B is not located in any of the granules considered in proximity (i.e., the ones included in S). The resulting uncertainty region of B, in this case, is equal to the entire space domain minus the region identified by S. When B is reported to be in proximity, A learns that B is located in one of the granules of S, but not exactly in which of those granules. Therefore, the uncertainty region in this case is given by the region identified by S. The size of this region depends on the value δ_A, on the area of the granules in G_B, and on the distance semantics chosen by A. In order to show how the size of the uncertainty region is affected by these parameters, we simplify the analysis by considering grid-like granularities, similarly to Section 5.2. Each granularity is a grid identified by the size l of the edge of its cells.

Figure 10 shows the additional privacy achieved by *C-Hide&Hash* for different values of δ/l. The additional privacy is measured as the lower bound of the number of granules in S. As can be observed, using both semantics, the additional privacy grows when δ is larger than l. This means, for example, that if δ is 5 times larger than l, then the actual size of the uncertainty region of B is 60 (or 88) times larger than the minimum privacy requirement if A is using the *maximum-distance* (or *minimum-distance*, resp.) semantics.

5.4 System costs

We separately evaluate the computation and communication costs involved in running the two proposed protocols. The analytical evaluation reported here is complemented with experimental results in Section 7.

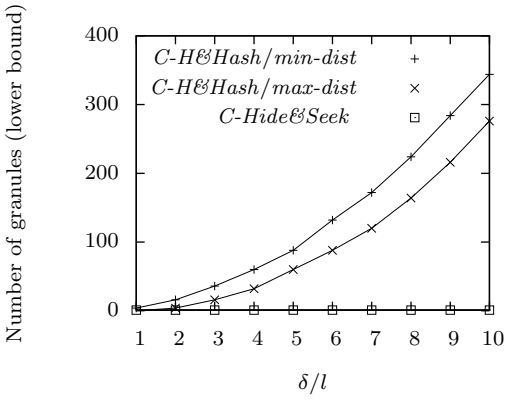


Fig. 10 Privacy with respect to a buddy

5.4.1 C-Hide&Seek

In order to perform a location update, a user needs to compute the index of the granule where she is located. The time complexity of this operation depends on the data structure used to represent granularities. As we shall show in Section 7, with our implementation of the granularities this operation can be performed in constant time. The complexity of the encryption operation depends on the encryption function and on the length of the encryption key. Considering a fixed key length, the encryption of the index of the granule can be performed in constant time. Since the SP only needs to store the received information, the expected computational complexity is constant. The communication cost is constant and consists in an encrypted integer value.

For what concerns the cost of a proximity request on the client side, for each buddy the issuing user needs to decrypt the index and to compute the distance of the granule with that index from her location. In our implementation these operations can be performed in constant time and hence the time complexity of the proximity request operation on the client side is linear in the number of buddies. On the SP side, the computational cost to retrieve the last known locations of the buddies is linear in the number of buddies. The communication consists in one request message of constant size from the user to the SP, and of one message from the SP to the user with size linear in the number of buddies.

5.4.2 C-Hide&Hash

The cost of a location update operation on the client is similar to the cost of the same operation using *C-Hide&Seek*, since the only difference is that a hashing function, which can be computed in constant time, is applied instead of the encryption function. Like in *C-*

Hide&Seek, the SP only needs to store the received information. Hence, computational costs of a location update are constant both for the client and for the SP. The communication cost is constant, as the only exchanged message consists in a hashed value.

On the client side, a proximity request from A requires, for each buddy B, the computation of the granules of G_B which are considered in proximity, the hashing, and the encryption of a number of granule indexes in the order of $sMax(G_B, \delta_A)$. The value of $sMax$ can be pre-computed for a given granularity. The computation of the granules considered in proximity can be performed in constant time in our implementation, using grids as granularities. The computation of the hashing and the encryption functions can also be performed in constant time, hence the time complexity of a proximity request is linear in the number of buddies times the maximum among the $sMax$ values for the involved granularities. When the client receives the response from the SP, the result computation performed by A for each buddy B requires the encryption of a number (the encrypted value sent by the SP), and the lookup of the encryption in a set of encrypted values with cardinality $sMax(G_B, \delta_A)$. As the lookup in the set of hashes requires at most $sMax$ operations, the time complexity is then linear in the number of buddies times the maximum value of $sMax$. Hence, this is also the overall complexity on the client side. On the SP side, the response to a proximity request from a user A requires, for each buddy B, a) the retrieval and the encryption of the hashed location of B, b) the encryption of the $sMax(G_B, \delta_A)$ hashed granule indexes sent by A. As the encryption runs in constant time, the time complexity is linear in the number of buddies times the maximum value of $sMax$.

Regarding the communication costs, both of the messages involved in the proximity request sub-protocol contain the encryption of a set of a number of hashed values linear in the number of buddies times the maximum value of $sMax$.

6 System implementation

We implemented the techniques presented in Section 4 in a system that provides proximity notification coupled with typical instant messaging (IM) functionalities. This implementation is the evolution of the system developed for the *Hide&Crypt* protocol and it has similar architecture, server and client applications [6].

The system is built as an extension of XMPP (Extensible Messaging and Presence Protocol), an open standard protocol often used in commercial applications as a message oriented middleware [15]. The system architecture is shown in Figure 11. The choice of extending XMPP is driven by the following considerations. First, the XMPP protocol can be easily extended to support custom services and messages, like the proximity service, in our case. In particular, by extending XMPP messages, we designed a proper XML protocol for each of our technique. In addition, the SP providing the proximity services is implemented as a XMPP component i.e., a pluggable entity that extends the default XMPP functionalities. A second advantage is that the XMPP protocol already includes standard sub-protocols for client-to-client communication and for managing the list of buddies. We used these sub-protocols as primitives in our implementation. Finally, since the XMPP architecture is decentralized, clients running on different servers can communicate with each other. In our case, since a component acts as a special type of client, this means that our proximity service is accessible to a user registered to an existing XMPP service, including popular IM services like Google Talk or Jabber. This makes it possible to use, in the proximity service, the same list of buddies used in those IM services. Clearly, proximity can be computed only for those buddies that are participating in the same proximity service.

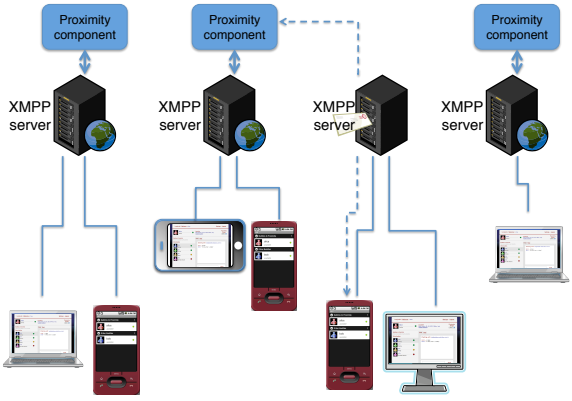


Fig. 11 System architecture

For what concerns the client, we developed a multiplatform web application and an other application specifically designed for Android based smartphones. In addition to the typical functionalities of an IM application, the clients implement the proximity protocols described in Section 4 and provide the typical functionalities of a full-fledged proximity service, including the detection of the client user's location, the notification of any buddies in proximity, and the graphical visualization of the location uncertainty region for each buddy.

One of the issues emerged during the implementation of the *C-Hide&Hash* and *C-Hide&Seek* protocols concerns key management. Indeed, both protocols re-

quire that each user A has a key K_A that is shared with all of her buddies, and it is kept secret to everybody else. A first problem is how A can share her key with one buddy B in a secure manner. This operation is required, for example, when the user accesses the proximity service for the first time or a new buddy is added to the buddy list. To address this problem, we employ standard public key cryptography techniques to encrypt, for each buddy of a user A, the key K_A; After being encrypted, the key can be safely transmitted over an insecure channel. The second problem is how to revoke a secret key. For example, this is necessary when a buddy is removed from the buddy list, or when the key is compromised. In our implementation, in order to revoke a key, it is sufficient to generate a new secret key and to send it to the authorized buddies.

The cost of sending a key to all the buddies is clearly linear in the number of buddies. In Section 7 we show that the costs to perform this operation on a mobile device are sustainable. In addition, it should be observed that the distribution of the key to all the buddies is only needed when a user first subscribes to the proximity service or when a buddy is removed from the buddy list. These are very sporadic events during a typical IM service provisioning.

7 Experimental results

We conducted experiments to measure the performance of our protocols and to compare them with the Pierre, `FriendLocator`, *Hide&Seek* and *Hide&Crypt* protocols [19,17,13]. We present the experimental setting in Section 7.1. Then, in Sections 7.2, 7.3 and 7.4 we evaluate the protocols according to three evaluation criteria: quality of service, privacy and system costs, respectively.

7.1 The experimental setting

The experimental evaluation of the protocols presented in this paper was performed on a survey-driven synthetic dataset of user movements, which was obtained using the *MilanoByNight* simulation[9]. We carefully tuned the simulator in order to reflect a typical deployment scenario of a proximity service for geo-social networks: $100,000$ potential users moving between their homes and one or more entertainment places in the city of Milan during a weekend night. The simulation also models the time spent at the entertainment places, i.e., when no movement occurs, following probability distributions extracted from user surveys. All the test results shown in this section are obtained as average values computed over $1,000$ users, each of them using the service during the 4 hours of the simulation. Locations are sampled every 2 minutes. The total size of the map is 215 km^2 and the average density is 465 users/km^2. All the components of the system are implemented in Java. Server-side test were performed on a 64-bit Windows Server 2003 machine with 2.4Ghz Intel Core 2 Quad processor and 4GB of shared RAM. Client-side tests were run on a HTC Magic mobile device, running Android as operating system. We implemented the symmetric encryption and the hashing functions using the RC4 and MD5 algorithms, respectively, while the RSA public key encryption algorithm was used for the key distribution.

In the experiments we used grid-based granularities. Each granularity is identified by the size of the edge of one cell of the grid. The location-to-granule conversion operations required by our protocol can be performed in constant time. For the sake of simplicity, in our tests we assume that all the users share the same parameters and that each user stays on-line during the entire simulation. Table 1 shows the parameters used in our experiments. Note that the "number of buddies" parameter refers to the number of *on-line* buddies that, for the considered type of application, is usually significantly smaller than the total number of buddies.

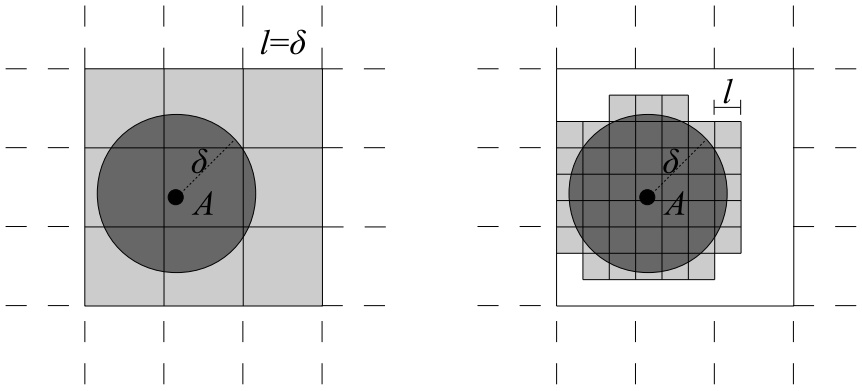
Table 1 Parameter values

Parameter	Values
δ	200m, **400m**, 800m, 1600m
Edge of a cell of G	100m, **200m**, 400m, 800m
Number of buddies	10, 20, **40**, 80

7.2 Evaluation of the quality of service

The first set of experiments evaluate the impact of the techniques on the *quality of service*, by measuring the exactness of the answers returned by each protocol. Indeed, two forms of approximation are introduced by our protocols. The *granularity approximation* is caused by the fact that, when computing the proximity between two users, the location of one of them is always generalized to the corresponding granule of her privacy requirement granularity. The other approximation, which we call the *time-dependent approximation*, is due to the fact that, when a user issues a proximity request with *C-Hide&Seek*, proximity is computed with respect to the last reported location of each buddy. The approx-

[9] `http://everywarelab.dico.unimi.it/lbs-datasim`



(a) Pierre / `FriendLocator` (b) *C-Hide&Seek*/ *C-Hide&Hash*

Fig. 12 Examples of the *granularity approximation*

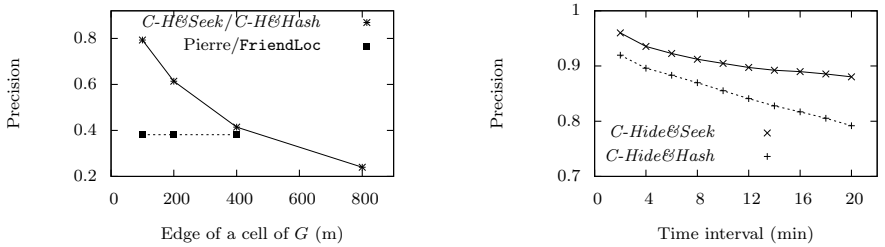
imation is introduced because the buddies have possibly moved since their last location update. Similarly, during the computation of a proximity request with *C-Hide&Hash*, the location transmitted by each buddy during the previous update interval is used.

For what concerns the *granularity approximation*, a similar problem occurs with the Pierre and `FriendLocator` protocols too. Indeed, both protocols, in order to detect proximity between buddies, partition the domain space into a grid, with each cell having edge l equal to the distance threshold δ, that must be shared by the users. Then, a buddy B is considered in proximity of A whether B is located in the same cell as A or in one of the 8 adjacent cells. The approximation introduced by these techniques depends entirely on the chosen value of δ. Differently, in our solutions, each user can choose her privacy requirements independently from the value of δ. For example, consider Figure 12. The black dot is the actual location of user A. The dark gray circle with radius δ is the area where the buddies of A are actually in proximity of A. The light gray area is the region in which buddies are erroneously reported to be in proximity[10]. Considering Figure 12(a), as l is always equal to δ when using Pierre or `FriendLocator`, the total area of the 9 cells considered in proximity is $9\delta^2$, while the area of the circle is $\pi\delta^2$, which is almost 3 times smaller. This means that, assuming a uniform distribution of the users, using Pierre or `FriendLocator` the probability that a buddy reported as in proximity is actually in proximity is about 1/3. On the contrary, in the protocols presented in this paper the size of the granules is independent from the chosen δ. In our example, this means that when the value l is smaller than δ, the region in which users are erroneously reported in proximity becomes smaller (Figure 12(b)).

[10] Here and in the following, we assume users of our protocols are choosing the *minimum-distance* semantics

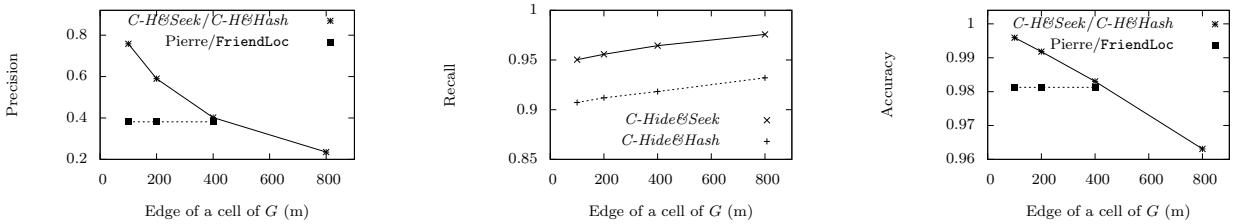
Figure 13(a) shows how the *granularity approximation* impacts on the service precision for different values of the edge of granularity cells. The metric we use for the measurement is the information retrieval notion of *precision*: the ratio between the number of correct "in proximity" answers over the total number of "in proximity" answers. Intuitively, the precision measures the probability that a buddy reported "in proximity" is actually in proximity. Note that the analysis would be incomplete without considering the notion of *recall*: the ratio between the number of correct "in proximity" answers over the sum of correct "in proximity" and incorrect "not in proximity" answers. Intuitively, the recall measures the probability that a buddy actually in proximity is reported "in proximity". In this case, since we are considering the *minimum-distance* semantics (see Section 5.2), the *granularity approximation* does not produce any incorrect "not in proximity" answer, and hence the recall is equal to 1. When conducting this experiment, in order to exclude from the evaluation the effects of the *time-dependent approximation*, for each buddy we used his current location as the last reported location. Since Pierre and `FriendLocator` do not consider G, their precision is constant in the chart and, as expected, is below 0.4. On the contrary, *C-Hide&Seek* and *C-Hide&Hash* have a significantly better precision when the edge of the cells of G is small. Intuitively, this is because the area where a buddy is erroneously reported as in proximity is smaller than δ (see Figure 12(b)). Figure 13(a) also shows the precision when the edge of a cell of G is larger than δ; The values are not reported for Pierre and `FriendLocator` since in this case they do not guarantee the privacy requirements.

Figure 13(b) shows the impact of the *time-dependent approximation*. The chart shows the results for our protocols only, as the other protocols proposed in the literature are not exposed to this kind of approximation.



(a) *Granularity approximation* only (b) *Time-dependent approximation* only

Fig. 13 Evaluation of the impact of the approximations



(a) Precision (b) Recall (c) Accuracy

Fig. 14 Evaluation of the quality of service (considering both approximations)

In order to exclude from this evaluation the effects of the *granularity approximation*, we performed these tests with the exact locations of the users, instead of the generalized ones. The chart shows, on the x axis, different lengths of the update interval and, on the y axis, the precision of the *C-Hide&Seek* and *C-Hide&Hash* protocols. It can be observed that *C-Hide&Seek* has better precision. This is due to the fact that *C-Hide&Hash* always uses the location reported during the previous update interval, while *Hide&Seek* uses the last location, that can be the one reported during the current update interval or during the previous one. Since the *time-dependent approximation* also introduces incorrect "not in proximity" answers, we also measured the recall. The corresponding chart is omitted as it is almost identical to the one in Figure 13(b). For example, using *C-Hide&Hash* and an update interval of 4 minutes, the value of the precision is 0.89 and the recall is 0.88.

The computation of the precision and recall under the *time-dependent approximation* confirms the intuition that using long update intervals negatively impacts on the quality of service. The choice of a value for the update interval should consider, in addition to this approximation, the cost of performing a location update. In general, the optimal value can be identified based on specific deployment scenarios. Considering our movement data, we chose 4 minutes as a trade off value since it guarantees precision higher than 0.9 and sustainable system costs as detailed in Section 7.3. Our choice is consistent with similar proximity services like, for example, Google Latitude that currently requires location updates every 5 minutes.

Figure 14 shows the analysis of the quality of service considering both the *granularity* and *time-dependent* approximations. Figure 14(a) shows the precision of our two protocols compared with the precision of Pierre and `FriendLocator`. We represent the precision of *C-Hide&Seek* and *C-Hide&Hash* with a single curve because the two protocols behave similarly. For example, when the edge of a cell of G is 200m, the precision of *C-Hide&Seek* and *C-Hide&Hash* is 0.59 and 0.57, respectively, while it is 0.61 for both protocols when the *time-dependent approximation* is not considered. This shows that this second type of approximation does not have a significant impact.

Figure 14(b) shows the recall of our protocols. Note that Pierre and `FriendLocator` do not lead to incorrect "not in proximity" answers, and hence their recall is equal to 1. On the contrary, our protocols can generate incorrect "not in proximity" answers due to the *time-dependent approximation*. This chart shows that the recall of *C-Hide&Seek* and *C-Hide&Hash* is always above 0.95 and 0.9, respectively. From Figure 14(b) we can also observe that the recall increases for coarser granularities. This is due to the fact that less incorrect "not in proximity" answers are returned if a coarser granularity is used. While this may appear unreasonable, the explanation is straightforward: there is an incorrect "not in proximity" answer only when a buddy is currently in proximity (considering Figure 12(b), his lo-

cation is in the dark gray area) while the location used in the computation of the proximity is outside the light gray area. If a granularity is coarse, then the light gray area is large and hence incorrect "not in proximity" are less frequent.

Figure 14(c) shows the *accuracy* for each considered protocol, i.e., the percentage of correct answers. Also in this case, the accuracy of *C-Hide&Seek* and *C-Hide&Hash* is represented with a single curve, as the two protocols behave similarly. Comparing this figure with Figure 14(a), we can observe that the accuracy achieved by all the protocols is much higher than the precision. This is due to the fact that this metric also considers the correct "not in proximity" answers that are usually the most frequent answers, since the proximity query area determined by the distance threshold is usually much smaller than the entire space. Figure 14(c) shows that our protocols achieve better accuracy than Pierre and `FriendLocator` when the value of the edge of the granularity cells is smaller than δ. In particular, for our default values, the accuracy of both *C-Hide&Seek* and *C-Hide&Hash* is higher than 0.99.

7.3 Evaluation of the system costs

The second set of experiments evaluates the computation and communication costs of the different protocols. For the analysis of the Pierre protocol, we used the NearbyFriend[11] application, developed by the same authors, which integrates the Pierre protocol in a desktop IM application.

First, we consider the costs related to the location update sub-protocol. This analysis does not apply to existing solutions as location updates are only required by our centralized solutions. As analyzed in Section 5.4, the temporal complexity of computing a location update is constant in the number of buddies. In our implementation, the computation of each location update requires, on the client side, about half of a millisecond for both the *C-Hide&Seek* and the *C-Hide&Hash* protocols. Similarly, the communication cost is independent from the number of buddies and the payload of each location update message consists in few bytes. Considering the overhead caused by the XML encapsulation, the dimension of each location update is in the order of a few hundred bytes.

The computation time needed to run a proximity request on the clients is shown in Figure 15(a). Note that the values reported in this figure only consider the computation time required by the issuing user. Indeed, all the protocols require the SP (in case of centralized services) or the other buddies (in case of distributed services) to participate in the protocol, and hence to perform some computation. For example, in the case of *Hide&Crypt* and Pierre, the total computation time of a user's buddies to answer a proximity request issued by that user is about the same as the computation time required to issue the request. As observed in Section 5, the computation time of a proximity request is linear in the number of buddies. Figure 15(a) shows that *C-Hide&Hash* requires significantly more time with respect to *C-Hide&Seek*, especially when the number of buddies is large. For example, the time needed to issue a proximity request for 40 buddies is about 20 ms for *C-Hide&Seek*, while about 900 ms using *C-Hide&Hash*. The figure also shows that the computation times of *C-Hide&Hash* and *Hide&Crypt* are similar, with *Hide&Crypt* performing slightly better. This is due to the fact that in *Hide&Crypt* each of the $sMax$ indexes only needs to be encrypted, while in *C-Hide&Hash* it also needs to be hashed.

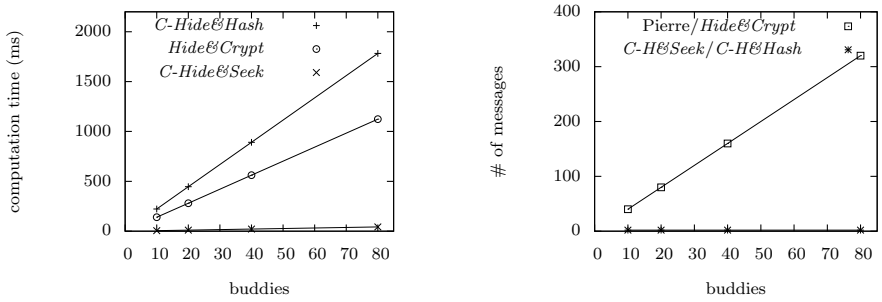
For what concerns other existing solutions, we did not implement the Pierre protocol on our mobile device platform. However, considering the experimental results presented by the authors (see [19]), the computation time of a single proximity request with a single buddy is more than 350ms[12]. Since, for *C-Hide&Hash*, the computation time on a mobile device of a proximity request with a single buddy is about 22ms, according to the data we have, our solution is at least one order of magnitude more efficient than the Pierre solution.

Regarding the computation costs on the server side, the complexity of a proximity request using *C-Hide&Hash* on the server side is similar to the one on the client side. However, in our experiments we observed that our high-end desktop machine is about 500 times faster than the mobile client to execute these operations. As a consequence, the computation for a single user having 40 buddies requires less than 2ms. While we did not run scalability tests on our server, this result suggests that, from the computational point of view, even a single desktop machine can provide the service to a large number of users.

Figures 15(b) and 15(c) show the system communication cost of a proximity request issued by a user. In Figure 15(b) we measure the number of messages exchanged by the system for each proximity request. It is easily seen that using a centralized protocol (i.e., *C-Hide&Seek* and *C-Hide&Hash*), only two messages need to be exchanged (one for the request and one for the response) independently from the number of buddies the issuer has. On the contrary, the decentralized protocols

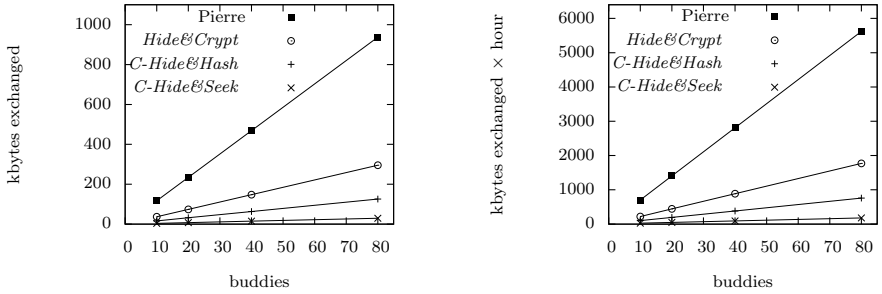
[11] `http://crysp.uwaterloo.ca/software/nearbyfriend/`

[12] It is unclear whether this result is obtained on a mobile device.



(a) Computation time to issue a proximity request

(b) Communication cost of a proximity request



(c) Communication cost of a proximity request

(d) Hourly communication cost generated by one user (location updates and proximity requests)

Fig. 15 Evaluation of the system costs

requires at least two messages for each buddy. Moreover, in our implementation of the *Hide&Crypt* protocol, each communication between two users needs to transit through the SP. The same applies to the Pierre protocol, using the NearbyFriend implementation. Consequently, at each location update, for each buddy, four messages transit in the system: two between the issuer and the SP and two between the SP and the buddy.

Figure 15(c) shows a comparison of the total amount of data exchanged in the system for each proximity request. Consistently with our analysis, the communication cost grows linearly with the number of buddies for both of our centralized protocols. It is easily seen that this also applies to the other protocols. The chart shows that NearbyFriend incurs in high communication costs. The reason is that, each time a proximity request is issued, a message of almost 3KB is sent from the user to each of her buddies and a message having a similar size is sent back in the reply. We believe that this overhead is mostly given by the fact that NearbyFriend needs all the communications between two users to be encapsulated in a secure channel. This is required because the Pierre protocol itself does not guarantee that any third party acquiring the messages cannot derive location information about the users. Since each message between two users transits through the server, the communication cost is almost 12KB for each buddy. The other decentralized solution we compare with, *Hide&Crypt*, has better communication costs. Indeed, each message is less than 1KB, and hence the cost is about 1/4 if compared to Pierre.

Our centralized solutions are even more efficient. This is due to the fact that only two messages need to be exchanged between the user and the SP for each proximity request. In case of *C-Hide&Hash*, each message has the same dimension than in *Hide&Crypt*, and hence, in this case, the communication cost is one half with respect to *Hide&Crypt*, and about one order of magnitude less with respect to Pierre. Finally, *C-Hide&Seek*, in addition to being a centralized solution, also benefits from the fact that each message contains only a few hundred of bytes. Consequently, this protocol is about 4 times more efficient than *C-Hide&Hash*.

In Figure 15(d) we evaluate the communication cost of the continuous use of a proximity service with our protocols. As mentioned in Section 7.2, we consider that location updates are issued every 4 minutes. Considering the results of our user survey, we use 10 minutes as the average frequency of proximity requests. The main difference of this figure with respect to Figure 15(c) is that it also considers the communication costs derived by the location updates. However, since each location

update costs less than 300 bytes, and 15 location updates need to be issued in one hour, the total hourly cost for this sub-protocol is about 4KB, which is negligible with respect to the communication cost of the proximity requests. The figure also shows that the centralized protocols require significantly less communication than the decentralized ones. In particular, *C-Hide&Seek* for one hour requires less than 100KB when the user has 40 online buddies. *C-Hide&Hash*, on the other side, requires 400KB per hour for the same number of buddies. We believe that this cost is largely sustainable on a wireless broadband network (e.g., 3G), and that, given the additional privacy with respect to curious buddies achieved using *C-Hide&Hash*, privacy concerned users may find this trade-off attractive.

Our experimental evaluation also included the measurement of the cost to distribute the private key (see Section 6). Both the computation and communication costs are linear in the number of buddies that need to receive the new key. For a single buddy, the computation time is about 7ms, measured on the mobile device, while the communication cost is less than 200 bytes. An experiment of key distribution to 40 buddies, resulted in a computation time of 275 ms, and a communication cost of 7KB.

7.4 Evaluation of the achieved privacy

In Section 5 we proved that both of our protocols guarantee the users' privacy requirements. We also observed that that *C-Hide&Hash* provides more privacy than what would be strictly necessary to guarantee the requirements. In this last set of experiments we evaluate how much additional privacy is provided by *C-Hide&Hash* in terms of the size of the uncertainty region. We recall that this is the area where a user A is possibly located as it can be computed by one of A's buddies after issuing a proximity request that returns A as in proximity.

Figure 16 shows that the privacy provided by *C-Hide&Hash* is always significantly larger than the privacy requirement, and it grows for coarser granularities G. Intuitively, with *C-Hide&Hash*, the uncertainty region corresponds to the union of the light and dark gray areas represented in Figure 12(b). Consequently, as the size of the cells of G decreases, the size of the light gray area tends to zero, and the uncertainty region becomes closer and closer to the dark gray area only. This means that the privacy provided by *C-Hide&Hash* is at least $\pi\delta^2$ even when the user requires her location to be obfuscated in a smaller area.

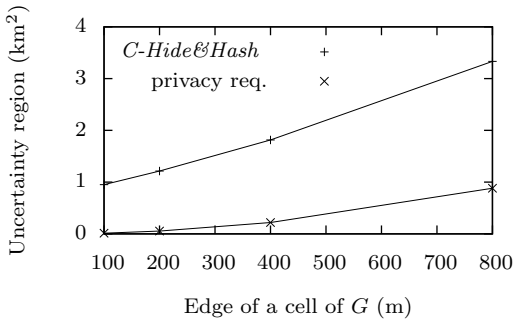


Fig. 16 Size of the uncertainty region.

8 Discussion and conclusions

We presented a comprehensive study of the location privacy problem associated with the use of a proximity service as an important component of any geo-social network. We illustrated two new protocols to compute users' proximity that take advantage of the presence of a third party to reduce the computation and communication costs with respect to decentralized solutions. We formally proved that the service provider acting as the third party, by running the protocol, cannot acquire any new location information about the users, not even in presence of a-priori knowledge of users' locations. We also showed that each user can have full control of the location information acquired by her buddies. Extensive experimental work and a complete implementation illustrate the benefits of the proposed solutions with respect to existing ones, as well as their actual applicability.

The two centralized solutions we propose require each user to share keys with her buddies, and hence are not the most appropriate to be used in a "query driven" service (e.g., finding people meeting certain criteria). The decentralized versions of the two presented protocols are more suitable in this case [13].

An interesting direction we plan to investigate is to extend the adversary models we considered in this paper to include not only (atemporal) a-priori location knowledge, but also *time-dependent* location knowledge. This would model not only a-priori knowledge about velocity, that our solutions can already deal with, but also a-priori probabilistic *proximity* information. It is still unclear if the proposed protocols, with appropriate location update strategies, similar to those discussed in Section 4.4, need to be modified in order to be proven privacy-preserving according to our definitions.

An interesting extension of our protocols is to allow users to specify different privacy preferences with respect to different groups of buddies. This is not difficult, but it exposes the users to dangerous collusion

attacks if further constraints are not imposed. The presented protocols are not subject to buddies' collusion attacks since each user defines the same granularity as privacy preference with respect to all of her buddies. If this is not the case, a user A, by assigning two different granularities with respect to buddies B and C to reflect her different level of trust, would expect that if B and C collude the lowest privacy requirement among the two is preserved. However, an adversary could actually intersect the uncertainty regions and potentially violate both privacy requirements. In order for our protocols to defend against such a collusion, some relationships need to be imposed on the granularities used in the system. While details are out of the scope of this paper, intuitively, granules from different granularities should never partially overlap. For example, using hierarchical grids as granularities would be a sufficient condition.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments and useful suggestions. This work was partially supported by Italian MIUR under grants PRIN-2007F9437X and InterLink II04C0EC1D, and by the National Science Foundation under grant CNS-0716567.

References

1. Arnon Amir, Alon Efrat, Jussi Myllymaki, Lingeshwaran Palaniappan, and Kevin Wampler. Buddy tracking - efficient proximity detection among mobile friends. *Pervasive and Mobile Computing*, 3(5):489–511, 2007.
2. Claudio Bettini, Sushil Jajodia, Pierangela Samarati, and X. Sean Wang. *Privacy in Location-Based Applications*, volume 5599 of *Lecture Notes in Computer Science*. Springer, 2009.
3. Claudio Bettini, X. Sean Wang, and Sushil Jajodia. *Time Granularities in Databases, Data Mining, and Temporal Reasoning*. Springer, 2000.
4. Matt Bishop. *Computer Security: Art and Science*, chapter 32. Addison-Wesley, 2003.
5. Hae Don Chon, Divyakant Agrawal, and Amr El Abbadi. Range and knn query processing for moving objects in grid model. *Mobile Networks and Applications*, 8(4):401–412, 2003.
6. Dario Freni, Sergio Mascetti, and Claudio Bettini. Hide&Crypt: Protecting privacy in proximity-based services. In *Proc. of the 11th International Symposium on Spatial and Temporal Databases*, volume 5644 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2009.
7. Gabriel Ghinita, Maria Luisa Damiani, Claudio Silvestri, and Elisa Bertino. Preventing velocity-based linkage attacks in location-aware applications. In *Proc. of ACM International Symposium on Advances in Geographic Information Systems*, pages 246–255. ACM Press, 2009.
8. Gabriel Ghinita, Panos Kalnis, Ali Khoshgozaran, Cyrus Shahabi, and Kian-Lee Tan. Private queries in location based services: Anonymizers are not necessary. In *Proc. of SIGMOD*, pages 121–132. ACM Press, 2008.
9. Marco Gruteser and Dirk Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. of the 1st International Conference on Mobile Systems, Applications and Services*, pages 31–42. The USENIX Association, 2003.
10. Panos Kalnis, Gabriel Ghinita, Kyriakos Mouratidis, and Dimitris Papadias. Preventing location-based identity inference in anonymous spatial queries. *IEEE Transactions on Knowledge and Data Engineering*, 19(12):1719–1733, 2007.
11. Ali Khoshgozaran, Houtan Shirani-Mehr, and Cyrus Shahabi. Spiral: a scalable private information retrieval approach to location privacy. In *Proc. of the 2th International Workshop on Privacy-Aware Location-based Mobile Services*. IEEE Computer Society, 2008.
12. Sergio Mascetti, Claudio Bettini, and Dario Freni. Longitude: Centralized privacy-preserving computation of users' proximity. In *Proc. of 6th VLDB workshop on Secure Data Management*, Lecture Notes in Computer Science. Springer, 2009.
13. Sergio Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, and Sushil Jajodia. Privacy-aware proximity based services. In *Proc. of the 10th International Conference on Mobile Data Management*, pages 31–40. IEEE Computer Society, 2009.
14. Peter Ruppel, Georg Treu, Axel Küpper, and Claudia Linnhoff-Popien. Anonymous user tracking for location-based community services. In *Proc. of the Second International Workshop on Location- and Context-Awareness*, volume LNCS 3987, pages 116–133. Springer, 2006.
15. Peter Saint-Andre. Extensible messaging and presence protocol (XMPP): core. RFC 3920, IETF, 2004.
16. Simonas Šaltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. *SIGMOD Rec.*, 29(2):331–342, 2000.
17. Laurynas Šikšnys, Jeppe R. Thomsen, Simonas Šaltenis, Man Lung Yiu, and Ove Andersen. A location privacy aware friend locator. In *Proc. of the 11th International Symposium on Spatial and Temporal Databases*, volume 5644 of *Lecture Notes in Computer Science*, pages 405–410. Springer, 2009.
18. Man Lung Yiu, Christian S. Jensen, Xuegang Huang, and Hua Lu. SpaceTwist: Managing the trade-offs among location privacy, query performance, and query accuracy in mobile services. In *Proc. of the 24th International Conference on Data Engineering*, pages 366–375. IEEE Computer Society, 2008.
19. Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, Lester and Pierre: Three protocols for location privacy. In *Privacy Enhancing Technologies*, volume LNCS 4776, pages 62–76. Springer, 2007.

A Proofs of formal results

A.1 Proof of Lemma 1

Proof The sought after independence intuitively means that whether an update/request is sent to SP by a user A is not related to where the user is located. Formally, by the definition of conditional probability, we have

$$\begin{aligned}
&P(ur(A)|loc_A, pri_A)\\
&= P(ur(A), loc_A|pri_A)/P(loc_A|pri_A)\\
&= (P(ur(A)|pri_A) * P(loc_A|pri_A))/P(loc_A|pri_A)\\
&= P(ur(A)|pri_A).
\end{aligned}$$

The second equality is due to the protocol, in which an update/request is sent at fixed time intervals for each user *independent* of the user's location. Hence, the lemma follows.

A.2 Proof of Lemma 2

Proof All we need is

$$P(M_1|M_2, loc_A, pri_A) = P(M_1|loc_A, pri_A),$$

i.e., the knowledge of the messages in M_2 does not have any impact on the probability of messages in M_1. But this follows the perfect secrecy assumption and the use of keystreams in our protocol.

A.3 Proof of Theorem 1

Proof We prove the theorem by showing that for each set M of messages exchanged during the protocol, we have $P(post_A) = P(pri_A)$. That is, the messages M do not change the SP's knowledge of A's location. By assumption of the theorem, $P(post_A) = P(loc_A|M, pri_A)$ as the only knowledge is M and pri_A. The knowledge that $loc_A \in g_A$ is useless as we assume in this case that g_A is the whole spatial domain. By the definition of conditional probability, we have

$P(loc_A|M, pri_A) =$
$P(M|loc_A, pri_A) * P(loc_A|pri_A)/P(M|pri_A)$.

It now suffices to show

$$P(M|loc_A, pri_A) = P(M|pri_A). \quad (2)$$

Intuitively, Equation 2 says that the messages M are independent of the location of A. This follows from two observations: the first is that the issuance of messages does not depend on the location of A by Lemma 1 and the second is that the (encrypted) messages are independent of the content of the messages by Lemma 2. More formally, assume

$$M = m_1, \ldots, m_n.$$

Let $ur(M)$ be the messages of the form

$$ur(m_1), \ldots, ur(m_n),$$

where $ur(m_i)$ is "an update/request is sent by user B_i". That is, $ur(m_i)$ disregards the encrypted part of the message but only says that a message is sent and by whom. By perfect secrecy assumption, the probability of a particular (single) message is the same as any other (single) message that differs only in the encrypted part, and hence the same as the probability of $ur(m_i)$.

Consider the case of two messages in M, i.e., $n = 2$. Now we have:

$$\begin{aligned}
&P\ (M|loc_A, pri_A)\\
&= P(m_1, m_2|loc_A, pri_A)\\
&= P(m_1|m_2, loc_A, pri_A) * P(m_2|loc_A, pri_A)\\
&= P(m_1|loc_A, pri_A) * P(m_2|loc_A, pri_A) \text{ by Lemma 2}\\
&= P(ur(m_1)|loc_A, pri_A) * P(ur(m_2)|loc_A, pri_A)\\
&\qquad \text{by the above discussion}\\
&= P(ur(m_1), ur(m_2)|loc_A, pri_A) \text{ by Lemma 2}\\
&= P(ur(M)|loc_A, pri_A)
\end{aligned}$$

The above can be extended to n messages in M and also to show the equation $P(M|pri_A) = P(ur(M)|pri_A)$. Hence,

$$\begin{aligned}
&P\ (M|loc_A, pri_A)\\
&= P(ur(M)|loc_A, pri_A)\\
&= P(ur(M)|pri_A) \text{ by Lemma 1}\\
&= P(M|pri_A)
\end{aligned}$$

and the thesis is established.

A.4 Proof of Theorem 2

Proof Given a buddy B, we prove the theorem by showing that for each set M of messages exchanged during the protocol, we have

$$P(loc_A|M, pri_A, loc_A \in g_A) = P(loc_A|pri_A, loc_A \in g_A),$$

where A is another user, and g_A is the location information that is encrypted in the messages of A with the key shared between A and B. In other words, we want to show that B will not acquire more location information about A through the messages other than what B already knows. Intuitively, this is true since the location information revealed by A is only at the granule level, but not *where* within the granule.

The formal proof is the same as for Theorem 1 but with the following two changes: (1) $ur(m)$ represents that request was sent from the granule included in the message if the message is intended to B; otherwise, it is the same as before. (2) $loc_A \in g_A$ is included in pri_A, or equivalently we replace each occurrence of pri_A with "$loc_A \in g_A, pri_A$". Let us now examine the steps in the proof of Theorem 1.

Lemma 1 still holds since updates/requests are sent regardless of locations if the user who sent the message is $C \neq A$. If $C = A$, then the $ur(A)$ gives the location (the granule) where the message is sent. In this case, the location is totally dependent on the given information of loc_A, $loc_A \in g_A$ and pri_A. Note that l is an index of a granule, any information contained in loc_A and pri_A below the granule level is not relevant to the probability of a message.

For Lemma 2, the content in M_2 still does not have any impact on the content in M_1 even when B can decrypt the messages intended to him as there is no information (from pri_A, loc_A, and $loc_A \in g_A$) that restricts any possible content in M_1, so the conditional probability of M_1 does not change regardless the existence of M_2.

For the discussion regarding the probability of m_i and $ur(m_i)$, with the addition of $loc_A \in g_A$, we still have that the conditional probability of m_i being the same as that of $ur(m_i)$. Indeed, assume

$m_i = \langle C, ui, E_{K^{ui}}(l)\rangle$.

If $C \neq A$, then all messages of the type have the same probability with or without knowing A's location since C's location information is not assumed in the conditional probability. This case is exactly the same as for the SP and the conditional probability of m_i is the same as that of $ur(m_i)$. If $C = A$, since B can decrypt the message, hence knowing the location l in the message, this location l (an index value of a granule in G_A) needs to be consistent with the location knowledge in loc_A and pri_A: if it is not consistent, then the probability of the message is zero; otherwise, the probability is totally dependent on the probability of A being in $G_A(l)$ given loc_A, $loc_A \in g_A$, and pri_A. But the same can be said about $ur(m_i)$ (which says that a message was sent at the given location), i.e., the probability of $ur(m_i)$ depends totally on loc_A, $loc_A \in g_A$, and pri_A. Therefore, m_i and $ur(m_i)$ have the same conditional probability. By the same reasoning as in the proof of Theorem 1, $ur(M)$ has the same conditional probability as M.

With all the above discussions, the theorem is established.

A.5 Proof of Theorem 3

Proof The proof follows the same style of that for Theorem 1. That is, we show $P(M|loc_A, pri_A) = P(M|pri_A)$, i.e., the location of A does not change the probability of messages M conditioned on pri_A. Like for Theorem 2, we examine the proof steps of Theorem 1 for the purpose of the current thesis. Lemmas 1 and 2 both hold due to the use of hashing function that displays stronger secrecy than encryption. The important difference is the discussion of the conditional probabilities of m and $ur(m)$. If m is an update, then the same applies as in the proof of Theorem 1. The difference is when m is a proximity request. In this case, the message contains multiple components. The critical step is to show that all such messages have the same conditional probability (to the SP) and hence the same as the conditional probability of $ur(m)$. This is not difficult since the location information in the condition is opaque to the SP. This opaqueness is given by two facts. The first is that the number of components in the message is the same regardless of the location information. The second is that the indexes of the granules and the "padding" (S'' in the protocol) in the message components are hashed and hence to the SP all possible granule indexes are equally possible in the encrypted (by K_1 in the protocol) message. (Here, hashing before encryption with K_1 is important as the adversary cannot attack using known pattern of the plaintext.) The above observations lead to the thesis of this theorem.

A.6 Proof of Theorem 4

Proof Intuitively, to the buddies, the *C-Hide&Hash* is much stronger than *C-Hide&Seek* since buddies only share a hashing function and the buddies location information is encrypted by a random key (generated by the SP) before sending to the requesting user B. Formally, the proof follows the same style as that for Theorem 2. The only difference is what it means when a message is "consistent" with the location knowledge. In this case, from B's perspective, we need to define $ur(m)$ to be the binary random variable that "the user is in one of the requesting granules or not" for the message sent back from the SP (as the reply to a proximity request from B). After B requesting proximity, B will receive a message from the SP with the encrypted hash value of A's location (in addition to the "kick back" from the SP in the form of encrypted values that B sent to the SP). Even though B and A shares the hash function, B does not know the encryption key which is randomly generated by the SP (K_2 in the protocol). Therefore, this value is probabilistically independent of the location of A. In this case, based on the protocol, the only information B obtains is whether A is in a granule among the ones given by B. This needs to be consistent with the location information contained in loc_A and pri_A. If not, then the probability of this message is zero, and otherwise the probability is totally dependent on loc_A and pri_A as no other information is available. The thesis follows the above discussions in the same style as the proof of Theorem 2.